Mini-project report 2 – graph-based recommendation engine

By: Martin Haver

1. Problem Statement:

The goal is to develop such recommendation engine that would satisfy the following scenario:

A user named Martin is going to browse on Amazon. The recommendation process is initiated by him landing on the home page. Amazon has already prepared a recommendation for a set of table tennis balls, because it knows, that last thing Martin bought was table tennis bat. If Martin will buy some of recommended items, the website will store this fact to the database and will be able to possibly infer new facts from this. The reason why the RS proposed table tennis balls is that it “knows” that many different users have bought them after buying table tennis bat, while rating the balls highly after the purchase.

The output of the process should be a ranked list of items based on what items the user has bought in the past and on what the other users have done. And since in this example the balls ended up first in the ranking, they were recommended to the user.

2. The data

The data used belongs among project SNAP datasets. It is based on “Customers Who Bought This Item Also Bought” feature of the Amazon website and was collected on March 02 2003. The data-set was subsequently processed to comply with required form *<IDA, IDB, nOfPurchases>.*

3. The solution

From this dataset, using Spark GraphX libraries, the graph is constructed in such a way that IDA (ID of item A) form an edge with IDB (ID of item B), nOfPurchases is its property, telling us how many times have been item B purchased right after product A by the users.

The algorithm

1. Randomly select a product by ID (to simulate the user has searched for a product with that ID)
2. Set it to be a starterNode
3. Find nodes connected to the starterNode
4. Count number of edges these endNodes have in common with the starterNode
5. Select 10 endNodes with the highest count of edges incoming from the starterNode
6. Recommend these endNode products in the given order

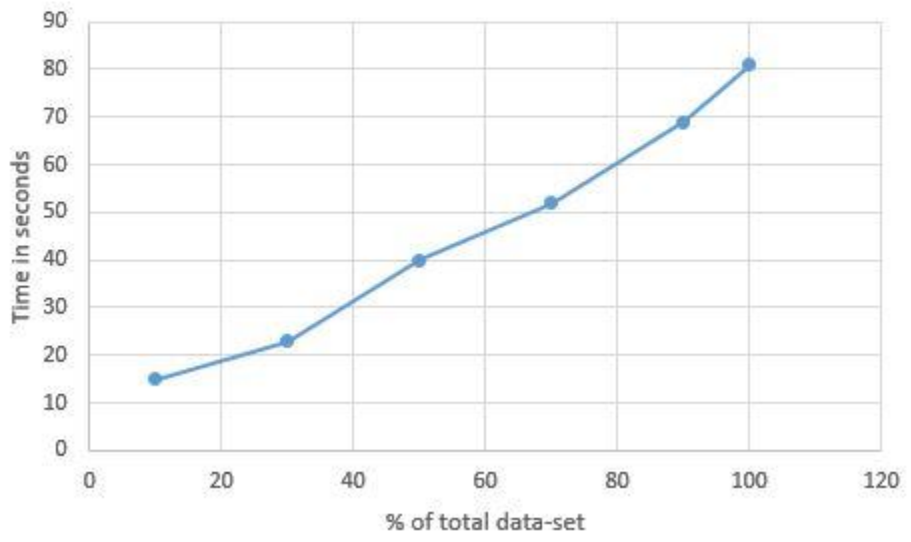
The higher number of users bought item A after item B, the higher the probability that this was not a random purchase and the items are logically related. And if thousands of such co-purchases occurred, it is almost certain that it was not just a coincidence, in accordance with the law of large numbers. My approach was inspired by one of Google's methods for ordering google query results that was presented in the class.

Item rating, for example in form of stars and sales rank (lower sales rank is, the better is the product selling compared to other products) can be then used to determine which product will be recommended in the end.

As development language was chosen Scala for its speed when running Spark programs, since Spark itself was also written in Scala. Spark GraphX used to create and process network of products was chosen out of curiosity about this technology and proved to be an effective way for processing data that can be represented in form of graph.

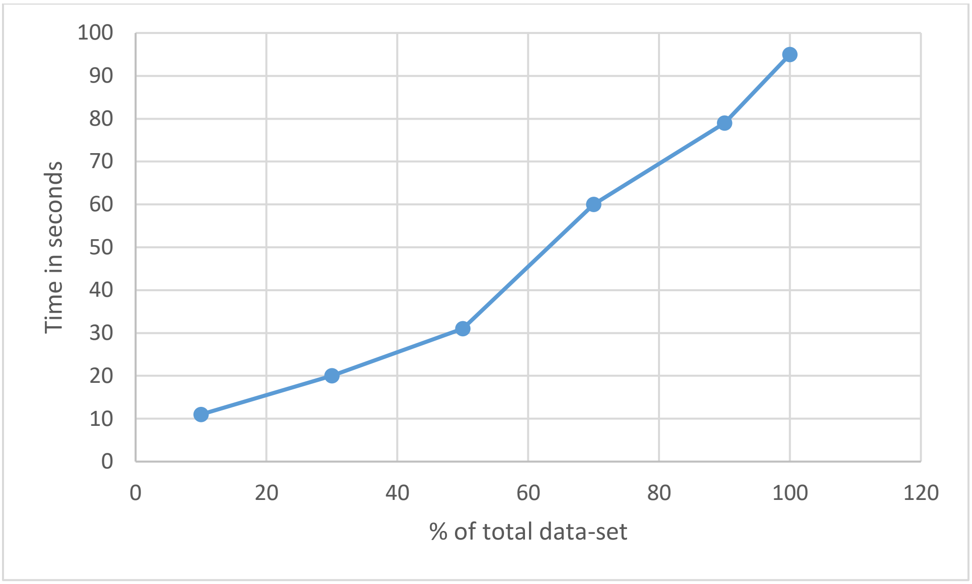
4. Experiments

The full data-set contained approx. 12 million pairs of products (207 megabytes in CSV format). This data-set was partitioned with different sizes to see how much will processing time increase with increasing amount of data. All tests were run 5 times and average result is shown here:



During experimentation significant effort went into trying various ways of optimization of the algorithm. Some of them significantly increased the processing speed. For example switching order of operations: ordering the graph edges by their nOfPurchases and filtering the graph for particular nodes (current order of operations – filter first, then order the rest) increased the speed by almost 90%, from 150 seconds to 80 seconds for the full data-set.

Another goal while experimenting was to show the improvement the parallel processing is offering. The original serial-working algorithm was adjusted by parallelizing the main filtering and sorting methods, which is one of the ways how to create RDDs (Resilient distributed datasets). RDDs are used in distributed computing to achieve theoretically faster, more efficient operations. However, I didn't manage to set the system up on a real, performing. All computations presented here were executed on my 5 year old notebook with i5 core processor and only 4 gigabytes of RAM. Therefore, the results have to be taken more as a hint on how the parallel way of computing the graph will perform, rather than an actual, objective test. The results of a parallel processing are here:



5. Conclusion

In order to improve the project and better understand the scalability of the algorithm, an actual cluster should be set up for this purpose, however this was out of scope of this project. Running on single machine, although in distributed fashion, can and probably will provide skewed results in terms of computing efficiency. However, especially serial approach has shown some potential and to conclude I believe that if exploited properly, graph-based approach to analysing similar data problems have bright future.